AD-A118 570 MASSACHUSETTS INST OF TECH CAMBRIDGE DEPT OF ELECTRI--ETC F/G 12/1
ALGORITHMS FOR PROGRAM VERIFICATION.(U)
MAY 82 V R PRATT AFOSR-80-0225

UNCLASSIFIED AFOSR-TR-82-0645 NL

END
DATE FILMED
9-82
DTIC

(12)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY AFOSR GRANT NUMBER 80-0225

May, 1982

Final Report

Algorithms for Program Verification

Principal Investigator: Vaughan R. Pratt

Reporting Period: July 1980 - June 1981

Abstract

In the previous reporting period, July 1979 to June 1980, work was performed on (1) foundations for specifying automatic program verifiers [Pr1,Pr3], (2) decision methods for a large fragment of the basic logic of programs [Pr5], (3) implementation of a decision method for propositional dynamic logic, and (4) continued maintenance of MITV1 [LP], a program verifier incorporating some of these ideas.

In the current (and final) reporting period, the research emphasis shifted from the verification of sequential programs to that of dataflow programs. This work has led to new insights into the correspondence between functions and processes [Pr4,Pr7]. It has also stimulated work on a new approach to user environments [Pr6]. And it has raised, though not answered, problems concerning automated verification of dataflow programs.

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DTIC
This technical report has been reviewed and is
approved for public release IAW AFR 190-12.
Distribution is unlimited.
MATTHEW J. KERPER
Chief, Technical Information Division

Research Objectives.

The overall objective of this project as originally conceived was to develop algorithmic tools for program verification, starting from prior foundations, developing whatever additional foundations were needed, and aiming ultimately at concrete implementations. In the course of the research it became evident that the emerging tools were equally applicable to other aspects of programs besides their verification. The logical algorithms developed for program verification by such workers as D. Oppen, J. Schwartz, R. Shostak, and the PI, were all useful for the compilation and optimization of programs. Such work is also likely to yield tools for the automatic analysis of programs, e.g. worst-case and average-case complexity analyses.

It was accordingly decided that it was intellectually uneconomical to continue to characterize the research as being purely for verification, and a broader objective was set, for which an appropriate title might be "Logical Algorithms for Program Analysis." The sense of the word "analysis" here is meant to embrace analyses yielding a wide variety of types of information, not by any means confined to properties expressing the correctness of the program.

Status of the research effort.

The two most important accomplishments during the reporting period were the development of a formal semantics for distributed computation, and the design of a programming environment based on that semantics. Both theoretical and implementation efforts are presently ongoing, well beyond the funding period of this grant (but presently unfunded as the PI of this grant is not yet eligible to be a PI at his new employing institution, Stanford).

Dataflow Semantics

In the previous report we outlined work then just beginning on dataflow semantics. This work is now well along, and has yielded a POPL paper [Pr7] and a presentation at a DARPA conference [Pr6].

Our theory closely parallels the ordered-pair theory of functions. Starting from a set P of ports and a set D of data values, we define a process event to be a port-datum pair, a process trace to be a partially ordered multiset of events, and a process (or process behavior) to be a set of traces. (The conventional notion of a function f could be defined in this way if we restricted the set P to be {I,O} (one input port I and one output port O), and required traces to be a totally ordered pair of events ((I,x),(O,f(x)), that is, an input event in which datum x arrives at port I and an output event in which datum f(x) leaves port O.)

We can define the semantics of networks of processes in an analogous fashion, substituting interprocess connections for ports to yield otherwise identical definitions for the notions of network event (a connection-datum pair), network trace (a partially ordered multiset of network events), and network behavior (a set of network traces).

Now define the restriction of a network trace to a process of the network to be the result of replacing connections in events of that trace by corresponding ports of that process to yield a process trace. We say that a network realizes a network process when for every process p of the network and every trace t of the network behavior, the restriction of t to p is a process trace of p.

By identifying certain processes of a network as its inputs and outputs, we can define the process realized by a network to be the restriction of the

network process realized by the network to its input and output processes. This passage from network behavior to process behavior is called abstraction.

We may now introduce the notion of a net-definable operation on processes. An n-ary net schema is a network of n uninterpreted processes numbered from 1 to n. Such a schema defines an n-ary operation mapping an n-tuple of processes to a process, namely to the process obtained by abstraction when the n-tuple of processes instantiates the n uninterpreted processes of the schema.

A net algebra is a set of processes closed under all net-definable operations.

Net theory consists of all valid equations between pairs of terms each involving net-definable operations along with variables ranging over processes.

In [Pr7] we raised three central open questions about net algebra and net theory. First, is there a finite basis for net algebra? That is, can all terms of net algebra be expressed using only a fixed finite set of net-definable operations? Second, what is the computational complexity of net theory? (Is it recursive, decidable in exponential time, etc?) Third, is net theory finitely axiomatizable?

The first question is important in formulating the remaining questions. In particular, the lack of a finite basis makes it harder, though not impossible, to formulate finite axiomatizability.

The second question is of central importance in any development of algorithms for reasoning about dataflow programs. If the theory has reasonable computational complexity then it can be used directly in logics of concurrent programs. If not, then alternative theories must be sought.

One might consider the third question to be of even greater importance for a theory in which proofs may be carried out. However in our opinion finite axiomatizability of a theory is too dependent on the form of the language to be a computationally interesting issue, and is mainly of academic interest.

These questions about parallel processes have their analogues in the corresponding questions for sequential processes. An adequate model for sequential processes, at this level of abstraction, is provided by finite automata, whose graphs supply the sequential analogue of parallel networks. Kleene (1967) in effect showed that three graphs supplied a complete basis for all graph-definable operations on languages. There is an obvious exponential-time decision procedure for equality between terms in such graphs: translate each side to its minimal deterministic automaton and then see if the same automaton is arrived at from each side. Redko (1967) showed that the equational theory of regular expressions is not finitely axiomatizable. (This does, however, raise the question as to whether for every finite basis the corresponding theory is not finitely axiomatizable; to the best of our knowledge this question has not even been raised, let alone answered, to date.)

Processes as a Basis for a Programming Language

In the course of developing a semantics for processes the possibility emerged of using processes as the conceptual basis of a programming language, paralleling Lisp's use of (recursively defined) functions for its conceptual basis, and Hewitt's [HB] use of "actors" for parallel processes.

In Hewitt's approach, "everything" is an actor. This "monotheism" is carried over to our process-based language, in which all data types have a common root, the process. Unlike Hewitt's actors, but like Lisp's functions (at least in theory), our processes are extensional in the sense that two processes with the same behavior are by definition the same process.

The extensionality property leads to some substantial simplifications of both the foundations and the implementation. A logic of processes can take the form of an algebra of processes, in which one asserts equalities between expressions denoting processes. Such an algebra is both natural and easy to develop foundations for. It is natural because it is the style of logic one encounters earliest in one's mathematical education, and it is suited to foundational work because there are many metatheoretical tools available from universal algebra for the study of equational theories of classes of algebras.

On the implementation side, extensionality eliminates the need for lazy evaluation as a semantic concept in its own right. In its place one may talk explicitly about say the set of even numbers or the set of primes. In this model lazy evaluation is just one of many possible implementations, and not generally a very good one at that.

This work will be submitted to POPL 83 as [Pr8].

Graphical Representation of Distributed Computation

Just as Lisp is syntax-free, or as Lisp's inventor McCarthy puts it, uses abstract syntax, so is our process language independent of syntax. This is not however to endorse the lack of syntax in programming languages. We have recently been giving some thought as to the most appropriate syntax for the above programming language.

Our present approach is to use all three dimensions for both input and output in all user interaction with the language. Spatial metaphors are used liberally to represent various concepts. Perspective projection is relied on to organize very large data bases: by keeping most data far away and only bringing currently interesting data near, one can keep an entire data base in view and yet have an uncluttered screen. Hierarchies are represented by physically grouping siblings together and permitting them to be manipulated as a unit. A list is represented as a linear arrangement of its elements. An infinite list may be displayed in its entirety on a screen with the help of perspective by having it disappear at a point at infinity, with most elements being vanishingly small. Dataflow programs are represented as physical networks through which data flows; even the flow of data itself can be represented naturally, in a way ideal for debugging programs, even in the special case where the programs are serial.

To manipulate this data, it is proposed to extend the traditional one-handed two-dimensional mouse to two hands, three dimensions, with grasping capability. The user waves the thumb and forefinger of each hand about in front of or within some sensing device (TV camera, sonar rangefinder, prosthetic device, etc.) and an image of these, modelled as two pairs of tweezers, appears on the screen. Each pair has 7 degrees of freedom, consisting of 3 for position, 3 for attitude, and 1 for grasping (thumb-forefinger separation).

A wide range of natural conventions is used to permit the user to communicate with the displayed data. Many operations on data are naturally modelled with physical movements of the tweezers. A datum may be selected by grasping it with one pair of tweezers. It may be dismantled by being pulled apart while grasping with both pairs. Two data may be assembled into a unit by grasping both and bringing them together. Menu items may take the form of buttons to be pushed. Analog quantities may be controlled as levers to be moved to and fro. Hierarchies may be searched by pulling data closer and later pushing it away. This only scratches the surface of the possible natural operations performable with tweezers.

This work is still in a very preliminary stage. We are anxious to find appropriate sources of support for a major thrust in this direction.

References

[BA] Brock, J.D. and W.B. Ackerman, An Anomaly in the Specifications of Nondeterminate Packet Systems, MIT LCS Computation Structures Group Note 33-1, Jan. 4, 1978.

[HB] Hewitt, C. and H.G. Baker, Laws for Communicating Parallel Processes, IFIP 77, 987-992; North-Holland, Amsterdam, 1977.

[KM] Kahn, G., and D. MacQueen, Coroutines and Networks of Parallel Processes, IFIP 77, 993-998, North-Holland, Amsterdam, 1977.

[Mil] Milner, R., Flowgraphs and Flow Algebras, JACM, 26, 4, 794-818, Oct. 1979.

[Opp] Oppen, D.C., Simplification by Cooperating Decision Procedures, ACM Trans. on Programming Languages and Systems, 2, 1, 1979.

[Pr1] Pratt, V.R., Axioms or Algorithms, Proc. 6th Symposium on Mathematical Foundations of Computer Science, Olomouc, Czech., Sept., 1979.

[Pr2] Pratt, V.R., Models of Program Logics, Proc. 20th IEEE Conference on Foundations of Computer Science, San Juan, PR, Oct. 1979.

[Pr3] Pratt, V.R., On Specifying Verifiers, Proc. 7th Ann. ACM Symp. on Principles of Programming Languages, Las Vegas, Jan. 1980.

[Pr4] Pratt, V.R., A Conceptually Economical Programming Language Based on Dataflow, unpublished manuscript, August 1980.

[Pr5] Pratt, V.R., Program Logic Without Binding is Decidable, Proc. 8th Ann. ACM Symp. on Principles of Programming Languages, Williamsburg, VA, Jan. 1981.

[Pr6] Pratt, V.R., Graphical Models of Distributed Computing, DARPA Conference on Graphical Representation of Software, Key West, Dec. 1981.

[Pr7] Pratt, V.R., On the Composition of Processes, Proc. 9th Ann. ACM Symp. on Principles of Programming Languages, Albuquerque, NM, Jan. 1982.

[Pr8] Pratt, V.R., A Dataflow Programming Language Based on Processes, to be submitted to the 10th Ann. ACM Symp. on Principles of Programming Languages, to be held Jan. 1983.

Publications

Pratt, V.R., Axioms or Algorithms, Proc. 6th Symposium on Mathematical Foundations of Computer Science, Olomouc, Czech., Sept., 1979.

Pratt, V.R., On Specifying Verifiers, Proc. 7th Ann. ACM Symp. on Principles of Programming Languages, Las Vegas, Jan. 1980.

Pratt, V.R., Modelling as a Paradigm for Verification, ACM SIGSOFT Software Engineering Notes, June, 1980. (Presented at VERKshop, SRI, April, 1980.)

Pratt, V.R., Program Logic Without Binding is Decidable, Proc. 8th Ann. ACM Symp. on Principles of Programming Languages, Williamsburg, VA, Jan. 1981.

Pratt, V.R., Graphical Models of Distributed Computing, DARPA Conference on Graphical Representation of Software, Key West, Dec. 1981.

Pratt, V.R., On the Composition of Processes, Proc. 9th Ann. ACM Symp. on Principles of Programming Languages, Albuquerque, NM, Jan. 1982.

Professional Personnel

Johann Makowsky, Ph.D. awarded 1974.

Derek C. Oppen, Ph.D. awarded 1974.

Vaughan R. Pratt (principal investigator), Ph.D. awarded 1972.

Michael Sipser, Ph.D. awarded 1979.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR- 82-0645	2. GOVT ACCESSION NO.: AD-A118570
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): ALGORITHMS FOR PROGRAM VERIFICATION (12)	5. TYPE OF REPORT & PERIOD COVERED: Final, 1 Jul 80–30 Jun 81
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Vaughan R. Pratt	8. CONTRACT OR GRANT NUMBER(s): AFOSR-80-0225
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Directorate of Mathematical & Information Sciences, Air Force Office of Scientific Research, Bolling AFB DC 20332	12. REPORT DATE: May 1982
	13. NUMBER OF PAGES: 6
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In the previous reporting period, 1 Jul 79–30 Jun 80, work was performed on (i) foundations for specifying automatic program verifiers [Pr1, Pr3], (ii) decision methods for a large fragment of the basic logic of programs [Pr5], (iii) implementation of a decision method for propositional dynamic logic, and (iv) continued maintenance of MITV1 [LP], a program verifier incorporating some of these ideas.

(CONTINUED)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED:

In the current (and final) reporting period, the research emphasis shifted from the verification of sequential programs to that of dataflow programs. This work has led to new insights into the correspondence between functions and processes [Pr4, Pr7]. It has also stimulated work on a new approach to user environments [Pr6]. And it has raised, though not answered, problems concerning automated verification of dataflow programs.